Basic - Libraries

EECS 201 Fall 2023

Submission Instructions

This assignment will be submitted as a repository on the EECS GitLab server. Create a private, blank, **README-less (uncheck that box!)** Project on it with the name/path/URL `eecs201-basic-lib` and add `brng` as a Reporter. The submission branch will be `lib`. If this branch is not already the default initial branch, you initialize the local repo with an additional argument: `git init --initial-branch=lib` if your version of Git is recent enough. Otherwise you can create a branch with this name after your first commit.

Preface

In this assignment you'll be provided yet another zipped archive containing some starter files.

`https://www.eecs.umich.edu/courses/eecs201/fa2023/files/assignments/basic-lib.tar.gz`

Extract the archive and `cd` into the created directory. **If you run `ls` you should see the `create` and `link` directories**.
Note that this assignment is to be submitted on a remote branch called `lib`. Initialize a Git repository **inside of the extracted `basic-lib` directory**; as noted above `git init -b <branch name>` or `git init --initial-branch <branch name>` can initialize the repo with a different branch name (e.g. `lib` as per the submission instructions). If your version of Git is too old for these options, you could create the `lib` branch afterwards after your first commit, or you can set the local branch's tracking information manually. Create a file called `report.txt` in this directory.

Add all of the present files and commit them.

Create a **private** project named `eecs201-basic-lib` on the UMich GitLab (`gitlab.eecs.umich.edu`) and add the instructor `brng` as a **Reporter**. Set this UMich GitLab project as your remote: you'll be pushing to it in order to submit.

This homework assignment assumes that you have a basic understanding of Makefiles. If you did not do Basic - Make or Advanced - Make, you may want to get brushed up on Makefiles.

1 Linking against libraries

In this section you'll be linking against a static library and its dynamic version with two different targets.

1. `cd` into the `link` directory.
2. Run the `getlib.sh` script. This script identifies your operating system and architecture to locate and create links to the appropriate provided library files, putting down the `libthingy.so` and `libthingy.a` symlinks in the `link` directory.
3. Open the `Makefile`. Note the two TODOs.
4. **As a foreword for Mac users, the `-l:libname.a` style of linking doesn't work with the default toolchain on macOS. If you want to link a static library, you can simply provide the path to the static library file as an argument, just as if it were a source code file e.g. `gcc -o file1.c libname.a`**.
5. Address the two TODOs and modify the compilation commands so that the `app-dynamic` target links against `libthingy.so` and the `app-static` target links against `libthingy.a`.
   When there's a naming conflict, which takes precedence?

6. **You can use the `run-dynamic` and `run-static` rules to see if you successfully linked against the dynamic and static libraries e.g. running `$ make run-dynamic`**. For Linux/WSL users, try running `ldd` on your executables to see if they dynamically linked the libraries. For Mac users, you can use `otool -L` to provide similar information about your executables.

2 Creating libraries

In this section you'll be performing the steps to build both a static library and a dynamic library.

1. `cd` into the `create` directory.
2. Take a look at the directory structure.
   - The `inc` directory contains library header(s).
   - The `libsrc` directory contains the source code for the library.
   - The `lib` directory (created by the Makefile) contains produced libraries.
3. Open the `Makefile`. Note the four TODOs.
4. Add the necessary commands to the appropriate recipes to build the object code for the dynamic and static libraries, the dynamic and static libraries themselves, and like in #1, update the `app-dynamic` and `app-static` targets to link against the appropriate libraries. For Mac users, remember that `-l:libname.a` style linker arguments don't work, and to provide the path to the static library as a normal file argument.
   - Note that dynamic/shared library objects need to be compiled as position-independent code.
   - Note that the object code has separate rules in the Makefile.

3 Conclusion

1. Add and commit the files you modified.
2. Fill out the `report.txt` file in the following steps:
3. On the first line provide an integer time in minutes of how long it took for you to complete this assignment. It should just be an integer: no letters or words.
4. On the second line and beyond, write down what you learned while doing this assignment. If you already knew how to do all of this, put down "N/A".
5. Commit your `report.txt` file and push your commits to your remote.
6. Run the autograder!